A top-down strategy to reverse architecting execution views for a large and complex software-intensive system

Callo Arias, Trosky B.; Avgeriou, Paris; America, Pierre; Blom, Krelis; Bachynskyy, Sergiy

Published in:
Science of computer programming

DOI:
10.1016/j.scico.2010.11.008

Publication date:
2011

Science of Computer Programming 76 (2011) 1098–1112

Contents lists available at ScienceDirect

Science of Computer Programming

journal homepage: www.elsevier.com/locate/scico

A top-down strategy to reverse architecting execution views for a large and complex software-intensive system: An experience report☆

Trosky B. Callo Arias[a,*], Paris Avgeriou[a], Pierre America[b], Krelis Blom[c], Sergiy Bachynskyy[c]

[a] University of Groningen, Nijenborgh 9, 9747 AG Groningen, The Netherlands
[b] Philips Research, High Tech Campus 37, 5656 AE Eindhoven, The Netherlands
[c] Philips Healthcare, P.O. Box 10000, 5680 DA Best, The Netherlands

ARTICLE INFO

Article history:
Available online 8 December 2010

Keywords:
Architecture reconstruction
Downstream software development
Execution views
Software-intensive systems

ABSTRACT

This article is an experience report about the application of a top-down strategy to use and embed an architecture reconstruction approach in the incremental software development process of the Philips MRI scanner, a representative large and complex software-intensive system. The approach is an iterative process to construct execution views without being overwhelmed by the system size and complexity. An execution view contains architectural information that describes what the software of a software-intensive system does at runtime and how it does this. The application of the strategy is illustrated with a case study, the construction of an up-to-date execution view for the start-up process of the Philips MRI scanner. The construction of this view helped the development organization to quickly reduce about 30% the start-up time of the scanner, and set up a new system benchmark for assuring the system performance through future evolution steps. The report provides detailed information about the application of the top-down strategy, including how it supports top-down analysis, communication within the development organization, and the aspects that influence the use of the top-down strategy in other contexts.

1. Introduction

Architectural information is an important asset for practitioners (e.g., architects and designers) for its potential to improve many aspects of the software development process, including the early detection of inconsistencies and undesired properties prior to implementation, better management, evolution, and reuse [1]. Architectural information is usually organized in multiple architectural views [2]. A view consists of one or more models, which are representations of a set of system elements and relations between them. A view and its models are developed conforming to an associated viewpoint [2]. A viewpoint frames particular concerns of the system stakeholders and consists of conventions and guidelines for the construction, interpretation, and use of a view. Often, in the case of large and complex software-intensive systems, architectural information is not well maintained or it may even never get to be documented. If system documentation is available, it may go into too much technical detail to offer relevant and accurate architectural information. As part of our research project on evolvability of software-intensive systems [3], we have observed that these situations are common in practice and development organizations constantly need up-to-date and accessible architectural information to steer the

☆ This work has been carried out as a part of the Darwin project at Philips Healthcare under the responsibilities of the Embedded Systems Institute. This project is partially supported by the Dutch Ministry of Economic Affairs under the BSIK program.
* Corresponding author.
E-mail addresses: trosky@cs.rug.nl (T.B. Callo Arias), paris@cs.rug.nl (P. Avgeriou), pierre.america@philips.com (P. America), krelis.blom@philips.com (K. Blom), sergiy.bachynskyy@philips.com (S. Bachynskyy).

0167-6423/$ – see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.scico.2010.11.008

development and maintenance of systems of this type. In addition, we have observed that the realization of a software-intensive system makes available a number of sources from which up-to-date and accessible architectural information can be reconstructed.

Architecture reconstruction is defined as the form of reverse engineering in which architectural information is reconstructed for an existing system [4]. Architecture reconstruction solutions are popular and successfully applied to extract architectural information about the *module* or *implementation* structure of software systems [4–7]. However, the extraction of architectural information about the *runtime* structure and behavior is still not very well supported [8], especially in the case of large software-intensive systems. This information is needed to construct execution views, which describe what a software system does at runtime and how it does it [9]. Execution views are organized into as-is and to-be models. While an as-is execution model describes the runtime of the current realization of an existing system, a to-be execution model describes the expected runtime of a system or feature to be implemented.

Constructing execution views for large software-intensive systems is a challenging task due to their complexity. Software-intensive systems combine various hardware and software elements, which are typically associated with large development investments and multidisciplinary knowledge. A particular characteristic of software-intensive systems is that their software elements take a considerable fraction of the development effort. These software elements are often as large as millions of lines of code, written in several different programming languages, and influence the design, construction, deployment, and evolution of the system as a whole. Over the last three years of our research project, we have developed and validated an architecture reconstruction approach [10,11] studying a Magnetic Resonance Imaging (MRI) scanner, a representative large and complex software-intensive system developed by Philips Healthcare [12].

The main benefit of our architecture reconstruction approach is that it enables the construction of execution views with high-level information at first and then, if it is needed by the stakeholders, detailed information to zoom in on areas of special interest. This helps practitioners, e.g. architects and designers, to construct execution views without being overwhelmed by the size and complexity of the system. Our experience developing and validating the approach aligns with the characteristics already identified by the research community (e.g. see [4,7]). In particular, we experienced that reconstructing execution views requires an iterative and tool-supported process, which must be customized according to the characteristics of the system at hand. In addition, we observed that the characteristics of the development process and the organization must be taken into account to put in practice, i.e. use, reuse, and embed, architecture reconstruction solutions.

In this article, we report our experience applying a top-down strategy to embed our architecture reconstruction approach in the incremental development process of the Philips MRI scanner. While our architecture reconstruction approach is a solution to construct up-to-date execution views, the strategy is a detailed process that describes which resources and steps are needed to run architecture reconstruction activities and use their results to support the incremental development of an existing software-intensive system. The strategy and its application are illustrated with a case study, which we conducted to document and analyze the start-up process of the Philips MRI scanner. The benefit of the case study included the identification of opportunities to quickly reduce about 30% the start-up time of the scanner, and the setting of a system benchmark to monitor the system's performance of future evolution steps.

Our report conveys three key aspects to support the reverse architecting of the runtime of an existing large and complex software-intensive system. First, it shows how to define the goal and means to steer the (re)construction of an up-to-date execution view. Second, it shows how to use an execution view during top-down analysis and downstream communication in a large development organization. Third, it shows how the systematic (re)construction of execution views can lead to improve and monitor quality attributes, such as performance. Together, these aspects show how the top-down strategy enables the use, reuse, and embedding of our architecture reconstruction approach within the incremental development processes of a large and complex software-intensive system.

The organization of the rest of this paper is as follows. Section 2 describes the motivation and details of the case study. Section 3 provides an overview of the architecture reconstruction approach. Section 4 presents the top-down strategy and its application in the case study. Then, Section 5 summarizes the technical contribution of the case study. Section 6 discusses related work. Section 7 discusses some aspects and open issues that influence the application of the top-down strategy. Finally, Section 8 provides some conclusions.

2. Case study: The start-up of the Philips MRI scanner

In this section, we describe the importance of the start-up process as a case study.

2.1. The start-up process

The start-up is a representative feature of the runtime and performance of software systems. Most development organizations continuously try to keep it within acceptable limits because of its technical complexity and relevance for end-user acceptance. The start-up process of a large software-intensive system is characterized by a large and complex set of interactions, e.g., communication and synchronization between software elements and hardware elements. This set of interactions is really an integration process that brings the system step by step to higher operational levels [13]. A typical integration process will create a higher operational level using available support functions from lower levels. An efficient

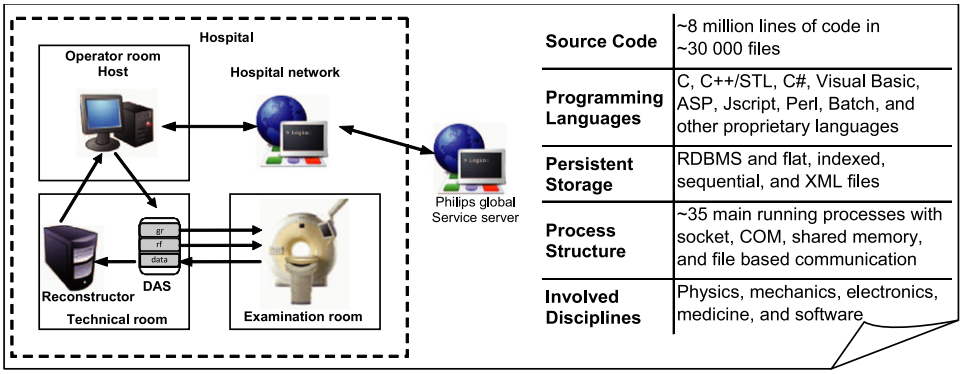

Fig. 1. Size and complexity profile of the Philips MRI scanner.

organization of this integration process is important to cope with exceptions (e.g., network not available), and mitigate the impact of failures and crashes [13], but also to achieve short start-up times. Since this is a design and architecture matter, development organizations need explicit and up-to-date descriptions to evaluate how to integrate additional system elements and to understand how changes in the current system elements and their interactions influence the start-up process.

2.2. The start-up of the Philips MRI scanner

The Philips MRI scanner is a representative large and complex software-intensive system in the healthcare domain. This system makes use of the physical phenomenon of magnetic resonance to obtain images from inside the human body in order to enable medical diagnosis. Fig. 1 summarizes the topology and implementation technology of the Philips MRI scanner, which illustrates its size and complexity. This also shows the number and complexity of the elements that need to be integrated as part of the start-up process that brings the Philips MRI scanner to the operational level. The topology in Fig. 1 illustrates that in the field (hospital) the system is distributed across three rooms, where three main computers identified as Host (end-user interface), Reconstructor (image reconstruction subsystem), and DAS (data acquisition subsystem) are connected to the system machine in the examination room. The system machine is specialized hardware that combines a superconducting magnet, a gradient system, and a radio frequency system. Other computers, e.g., the Hospital network and Philips Global Service server are less important in this context since the interactions with them do not influence the start-up performance significantly.

The start-up time of the Philips MRI scanner, the time to bring the system to an operational level, is measured as the period between the event that the end-user presses the power-on button of the Host computer and the event that the system allows the end-user to input the data to start a medical diagnosis. In this period of time, the software elements, deployed inside the three main computers, are loaded to build a distributed runtime process structure (see Fig. 1). In this structure, software elements are mapped to runtime processes that interact with each other using a number of communication mechanisms and technologies. Each runtime process loads and runs binaries built with the system source code, about 8 million lines of code. For many years, the start-up process was not perceived as a hot issue and most of the knowledge about it was distributed over the minds of several practitioners. These practitioners were not sure about the validity of their knowledge, as well as the efficiency of the current characteristics of the start-up process, especially due to the considerable changes in the system over the recent years. For this reason, the development organization, represented by the software architecture team, decided to document the current state of the start-up process to prevent possible future major issues.

2.3. Case study concerns and decisions

The goal of the development organization was to make explicit the actual organization of the start-up process and its duration. Achieving this goal will provide the organization with accurate information to address a set of concerns expressed by the following questions:

- What are the actual elements involved in the system start-up?
- What are the aspects that constrain, coordinate, and control the runtime concurrency of the system start-up?
- What are the bottlenecks and delays, if any, of the system start-up and their root causes?
- How do the actual start-up characteristics match the current knowledge, i.e., mental models of practitioners?
- What are the opportunities for improving the system start-up?

Since the set of concerns were related to the runtime of the system, the development organization decided to address them by constructing execution views with the architecture reconstruction approach summarized in Section 3. The selection

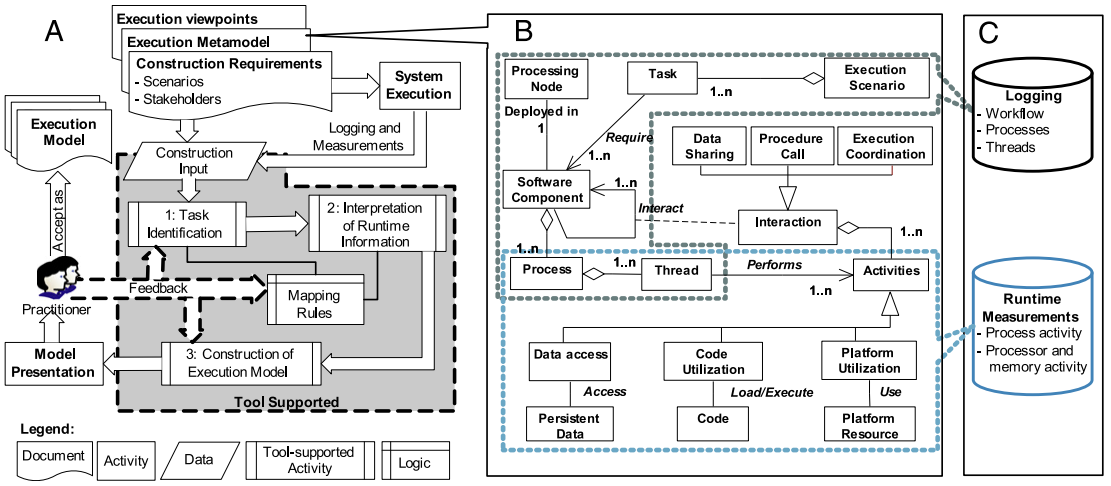

Fig. 2. Overview of the architecture reconstruction approach.

Table 1
Summary of execution viewpoints.

Stakeholders: Software architects, designers, developers, testers, and system platform supporters.
Development activities: System understanding, analysis of alternative designs and implementations, introduction of new hardware resources, testing, conformance of design and implementation, corrective maintenance.

Viewpoint	Concerns	Model types
Execution profile	- What are the major components that realize a given system function? - What are the high-level dependencies of major components? - What are the major tasks that build the execution workflow of key execution scenarios? - What is the development team that develops or maintains a given system function?	Functional mapping, Execution workflow, Matrix model, Sequence diagrams
Execution concurrency	- Which runtime elements execute concurrently? - How does the runtime concurrency match the designed concurrency? - What are the aspects that constrain, coordinate, and control the system's runtime concurrency? - What are the bottlenecks and delays of the system and their root causes? - What are the opportunities to improve the concurrency of the system?	Workflow concurrency, Process and thread structure, Control and data flow

of the approach was driven by two factors. First, it was shown that the approach could build useful execution views at a high-level of abstraction without touching or knowing the large and complex system implementation. Second, the cost of training or preparation to use the approach can be neglected. This was possible because the approach was developed and validated in close collaboration between researchers and Philips Healthcare MRI practitioners as part of the Darwin project [3], an industry-as-laboratory research project.

Finally, we designed a top-down strategy and assembled a team. The strategy, core of this experience report, is presented in Section 4. Two practitioners (a software architect and a designer) and a researcher composed the team. The role of the software architect was to document the start-up process and coordinate development activities to improve the system start-up. The designer played the role of technical expert of the system start-up. The researcher guided the application of the strategy and the architecture reconstruction approach.

3. The architecture reconstruction approach

Fig. 2 presents an overview of our architecture reconstruction approach for execution views. The overview illustrates that the approach put together an iterative and tool-supported process (A), a set of viewpoints and a metamodel (B), and sources of runtime information (C). The execution of the approach enables the construction of as-is execution models. In this section, we introduce the elements of the approach describing the input, steps, and output of the iterative process.

3.1. Inputs to the process

The set of execution viewpoints, the execution metamodel, the source of runtime information, and a description of the construction requirements represent the Construction Input for the process, see A in Fig. 2. The execution viewpoints frame a set of concerns regarding the runtime behavior and structure of software-intensive systems. These viewpoints provide reusable guidelines to construct and use execution profile, execution concurrency, and resource usage views [9,14]. In Table 1, we summarize two of the execution viewpoints that frame concerns like the ones held by Philips Healthcare MRI

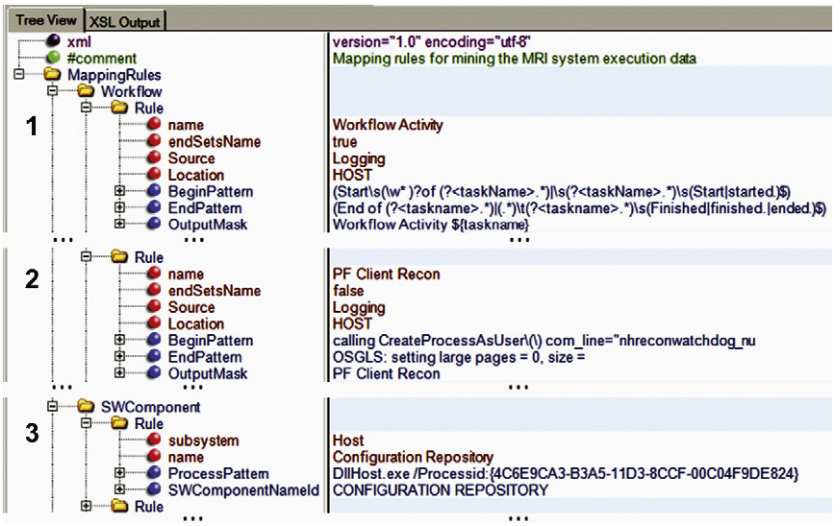

Fig. 3. Examples of mapping rules.

regarding the start-up process, see Section 2.3. The summary includes the stakeholders that hold the concerns, the development activities where the concerns often arise, and the model types used to present architectural information that address the concerns. Further details about the execution viewpoints and the model types are documented in [14]. In Section 4, we describe further details about the construction and use of execution workflow and workflow concurrency model.

The execution metamodel, illustrated by B in Fig. 2, provides a set of elements and relationships that can be used to describe the runtime structure and behavior of a specific software-intensive system. The elements in the execution metamodel can be extended or specialized according to the problem at hand, e.g., Fig. 7 shows additional relations for the Task concept that are required for the case study. The links between B and C in Fig. 2 are to illustrate that instances of the elements and relationships in the execution metamodel can be extracted from sources, such as logging and runtime measurements. For our approach, logging is collected using the built-in logging infrastructure of the system. Runtime measurements are collected using monitoring tools provided by the system runtime platform, e.g., the Sysinternals suite [15].

The construction requirements consist of a specification that includes at least two items. First, a set of execution scenarios that, under the criteria of the development organization, cover the execution of the feature or functionality related to the problem under analysis. Second, a list of stakeholders and their concerns. The stakeholders include the practitioners and other personnel of the development organization that need to be involved in the process as sources of support, information, validation, or ultimately as users of the constructed view.

3.2. Steps in the process

The steps in the process, illustrated by A in Fig. 2, are tool-supported activities, i.e., task identification, interpretation of runtime information, construction of the execution model, and model presentation. The first three activities implement a dynamic analysis technique that extracts and abstracts architectural information from the source of runtime information. The extraction of architectural information is a rule-based mechanism that uses a set of mapping rules. Fig. 3 gives examples of the definition of four mapping rules. The mapping rules are stored in a repository, an XML file, as a set of parameters that capture patterns about the syntax and semantic of the runtime data. For instance, the definition of the first mapping rule in the figure includes two parameters, BeginPattern and EndPattern. These parameters store text patterns that can be applied to characterize some logging messages as workflow activity, which identify the boundaries of the tasks or aggregations that build the workflow of an execution scenario.

The tool support for the technique includes a set of Python scripts embedded in a .Net application, which combines the sources of runtime information and applies the mapping rule mechanism. Further details about the implementation of the dynamic analysis technique are reported in [10]. More about mapping rules is described as part of the application of the top-down strategy in Section 4.

3.3. Output of the process

The output of running the dynamic analysis technique is an execution model that presents a subset of the extracted information. Figs. 6 and 8 are examples of execution models, which describes the runtime of a system using instances of the elements in the metamodel. Fig. 7 describes the specific runtime elements and notations for these kind of models. In this case, the types of these execution models correspond to execution workflow and concurrency model, defined as part of the

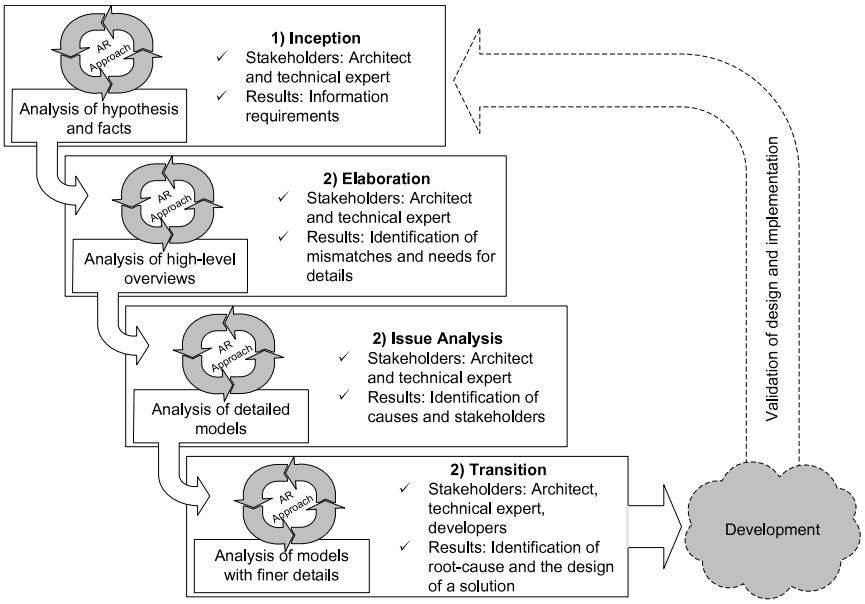

Fig. 4. Top-down strategy to reverse architecting execution views.

execution viewpoints. When an execution model is constructed, it is presented to the involved stakeholders as part of the model presentation activity.

The goal of the model presentation activity is to let the stakeholders, e.g. software architects and designers, check whether the model provides useful and sufficient information to conduct an analysis activity. If the feedback indicates that the constructed model needs to be fine-tuned, several iterations of the dynamic analysis and model presentation may be necessary, especially when a model is constructed for the first time. Otherwise, the model can be accepted as one of the models in the execution view that supports the analysis and solution of the problem at hand. For our approach, the tool support for the construction and presentation of execution models includes software tools such as Graphviz and Microsoft Excel.

4. The top-down strategy and its application

Fig. 4 presents an overview of the top-down strategy, which illustrates that the strategy involves four main phases, inception, elaboration, analysis, and transition. The names of the first, second and fourth phases match three of the well known Rational Unified Process (RUP) phases [16]. This naming reflects the similarities in goals between the RUP phases and the phases of the top-down strategy. The figure also illustrates that the phases in the strategy are successive individual iterations of our architecture reconstruction (AR) approach. These iterations aim at producing information that is required to achieve systematically the goal of the case study as follows:

- In the inception phase, the iterations aim at collecting facts and constructing a hypothesis about the runtime of the system, which can be further analyzed to define the scope of the problem. The scope of the problem includes the execution scenarios, stakeholders, and an initial set of information requirements that are required to steer the next phases.
- In the elaboration phase, the iterations aim at constructing an execution model that provides an up-to-date overview of the execution scenarios defined in the previous phase. The analysis of the overview should make it possible to address the initial set of information requirements. From this analysis, further information requirements can be derived, especially when the overview exposes issues, i.e., mismatches with the hypothesis or expectations of the stakeholders.
- In the analysis phase, the iterations aim at constructing execution models with details that describe better the issues exposed during the analysis in the previous phase. The analysis of the details in the execution models should make it possible the identification of the origin of the issue and the developers that can explain or fix it.
- In the transition phase, the iterations aim at constructing execution models with even finer detail information about the issues identified in the previous phase. The finer details in the models should help the developers, responsible for the issue, to find the root cause of the issue and design a solution to fix it.

The development phase is not a dedicated part of the strategy. After this phase, repeating any of the four phases of the strategy can be required, especially to verify and validate the design and implementation of solutions designed in the transition phase. In the rest of this section, we describe our experience applying each phase of the strategy in the case study.

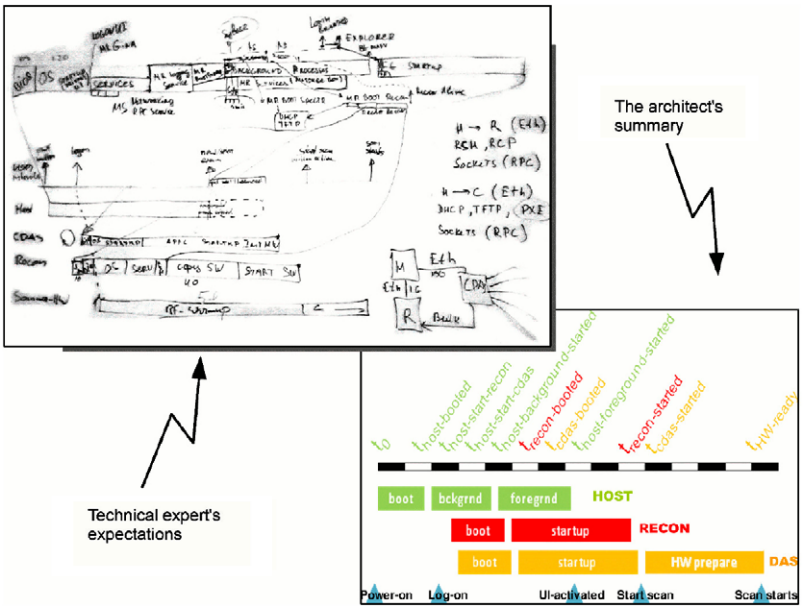

Fig. 5. A hypothetical view of the system start-up.

4.1. Phase 1: Inception

The challenge in the inception phase is to get the scope of the problem as clear as possible without the risk of analysis-paralysis. In order to achieve this, we start by identifying the stakeholders' concerns around the problem at hand. Then we identify the set of concerns that match the ones framed by the execution viewpoints. A positive matching will confirm that the problem at hand requires the construction of up-to-date execution views. At this stage, the first iteration of the architecture reconstruction approach will focus on the completion of the other elements in the construction input (see Section 3.1), e.g., stakeholders and scenarios, following the guidelines of the identified viewpoint(s).

For the start-up case study, as described in Section 2.3, the stakeholders' concerns match the execution profile and execution concurrency viewpoints (see Table 1 and [14]). Then, we identified the execution scenarios and stakeholders. The start-up process of the Philips MRI scanner can be represented by different execution scenarios, e.g., some can include hardware crashes and others software upgrades. To focus the problem, we decided to look only at clean and stable scenarios without exceptions, crashes, or upgrades. To understand the most representative stable and clean scenario, we constructed a hypothetical view, illustrated in Fig. 5. This view describes the expected system behavior through a sketch and a high-level diagram created by the technical expert and the architect, respectively, following the guidelines of the execution profile and execution concurrency viewpoints.

The guidelines in the viewpoints suggest that to in order to cope with size and complexity, an execution scenario (e.g. the system start-up) can be described with a execution workflow or workflow concurrency model [14]. Fig. 7 describes the concept and notations used in these types of models. A model of any of these types presents an execution scenario as a set of tasks that build the scenario workflow. The set of tasks correspond to aggregations of the steps run by the end-user or automatically executed by the system to realize the corresponding system functionality or feature. In addition, if possible, the model describes how the identified tasks map to major hardware and software elements of the system. The details about other relationships, e.g. temporal relations, between tasks can be included if these are required for the problem at hand.

The sketch describes the tasks for the start-up scenario as the aggregations of events and steps, which according to the technical expert should happen to bring the system to an operational level. In addition, the expert included estimations about the duration of the identified tasks, their temporal relations, and communication mechanisms among them. To keep the scope of the problem and due to the organization of the team, the sketch was based only in the mental models of the technical expert. The architect summarized the information in the sketch by constructing the diagram in the hypothetical view. The diagram describes the start-up in terms of high-level tasks that are distributed over time and in the main computers of the system. The diagram includes the period of time and events that are perceived by the end-user, which according to the architect were not explicit and need to be investigated.

The analysis of the hypothetical view contributed with a number of aspects to the inception and continuation of the case study. First, it helped to share insights, e.g., hypothesis and facts, about the start-up process. For example, we identified the involved major hardware (e.g., Host, Reconstructor, and DAS computer), and how the software elements are organized and aggregated (e.g., Background and Foreground processes in the Host computer). Second, it helped to make explicit an initial set of information requirements that should be addressed (e.g., verify the estimations and assumptions about the duration

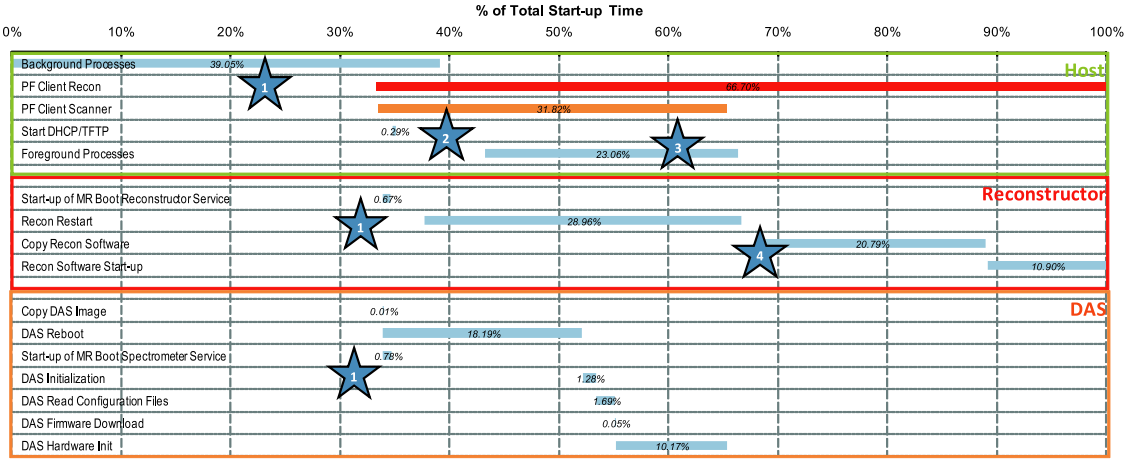

Fig. 6. An as-is overview model of the system start-up.

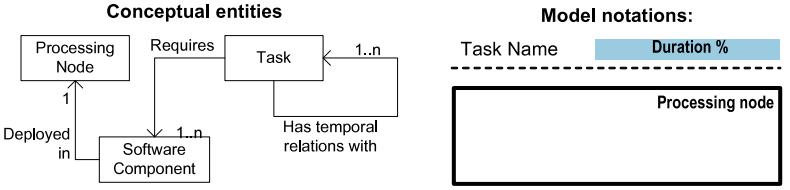

Fig. 7. Concepts and notations for execution workflow and concurrency models.

and distribution of the major tasks in the start-up process). Third, it helped to decide that a workflow concurrency model would be sufficient to start with the description of the actual system start-up process.

4.2. Phase 2: Elaboration

The challenge in the elaboration phase is to construct an as-is high-level description of the current system realization without being overwhelmed by its size and complexity. To achieve this, we focus on an initial set of information requirements, as identified in the inception phase. Then we apply the architecture reconstruction approach to construct as-is execution models that address the initial set of information requirements. This involves gathering the runtime data, definition of the required mapping rules, and execution of the dynamic analysis technique of the approach.

For the start-up case study, the initial information requirement was identify actual information about the boundaries and distribution of the main tasks that build the workflow of the system start-up. We started by collecting logging and runtime measurements from the start-up process of a stable system release. Then, we used the dynamic analysis technique to extract actual runtime information from the collected data. As we described in Section 3.2, the technique consists of three activities: task identification, interpretation of runtime information, and the construction of the execution model. In this phase of the strategy, task identification focused on the validation of mapping rules that enable the semiautomatic identification of the boundaries of the tasks or aggregations that build the workflow of the system start-up. Interpretation of runtime information focused on processing the collected runtime data to, at least, extract actual information about starting time and duration of the identified tasks, and their actual distribution across the system computers. Finally, the construction of the execution model focused on the construction of an overview, an as-is execution workflow model, following the guidelines of the execution profile viewpoints, just like it was done in the hypothetical view (Fig. 5).

For the construction in this phase, we tuned the dynamic analysis technique to address several practical aspects, especially to identify the boundaries of some of the tasks within the start-up process. This identification is done in the Task Identification activity of the approach, see Fig. 2 A, by using mapping rules. The identification is done selecting logging messages that match mapping rules for workflow activity, e.g. the first mapping rule in Fig. 3. However, the initial iteration in this phase showed that the implementation of the mechanism and the used mapping rule were not enough to automatically identify all the relevant tasks of the system start-up scenario. Thus, as part of the validation, the technical expert provided more details to extend the mapping rule repository with additional rules such as the PF Client Recon rule, see Fig. 3, which were required to identify missing tasks.

Extending the mapping rules, the technical expert pointed out additional patterns in the logging messages, which we incorporated to refine the major tasks into finer and still self-contained tasks. Fig. 6 shows the execution model that we constructed in this phase. The model corresponds to a execution workflow model type. Fig. 7 shows the concepts and notations for this type of execution model, which are also documented in the execution profile viewpoint [14]. The model

in Fig. 6 and the other models in the rest of this section are simplified version of the actual models in the case study. The absolute time values in the models are mapped to percentages of the actual start-up time value for publication purposes. The construction and presentation of this model and the other as-is models in the case study were done using Microsoft Excel.

In contrast to the diagram in the hypothetical view (Fig. 5), the execution model in Fig. 6 shows additional information, but also excludes some other. The included information is the actual time characteristics and the finer task decompositions of the execution scenario. We excluded the description of the boot task in the Host computer. The analysis showed that it was a constant factor that could be analyzed separately using tools such as Bootvis [17], which enable the capture and graphical display of boot and resume performance trace data in Windows XP.

In the analysis of the as-is workflow overview, we compared the expected time values, provided by the technical expert, with the ones in the overview. This analysis enabled the identification of four point of attention, see the numbered stars in Fig. 6.

1. The duration of the task Background Processes, on the Host computer, was larger than expected. As a consequence, the tasks PF Client Recon and PF Client Scanner (interfaces to the other two computers) were starting later than expected.
2. A gap between Background Processes and Foreground Processes task was not considered in the hypothesis. This gap represents the period of time that an end-user takes to manually trigger the loading of the main user interface in the Host computer.
3. The Foreground Processes can be faster, which can speed up the presentation of the main user interface.
4. Based on knowledge about the function of the Copy Recon Software task, its duration time should be shorter.

The conclusion of the analysis was that only 1, 3 and 4 represented issues and further investigation might lead to the identification of considerable opportunities for improving the system start-up.

4.3. Phase 3: Analysis

The challenge in the analysis phase is to select and recover detailed information that supports the analysis of mismatches and issues, identified in the previous phase, without being overwhelmed by the system implementation size and complexity. To achieve this, we construct as-is execution models that zoom in on the areas that contain the mismatches and issues.

For the start-up case study, this phase focused on the construction of an execution model to zoom in on the first and second issue, marked by the stars 1 and 3 in Fig. 6. We followed the guidelines of the execution profile viewpoint, which suggest that to dig down for details, a task can be represented with finer aggregations, e.g. subtasks. Following the metamodel (B in Fig. 2), we decided that the finer aggregation will correspond to the task that represent the start-up of the software components within the system start-up process. In addition, the metamodel shows that a software component represents a set of processes that belong together due to parent–child relationships, or shared functional or non-functional characteristics defined by the development organization. Therefore, we decided that the start-up of a software component is represented by an aggregation of the runtime activities performed to initialize the component's processes and their required resources.

The reconstruction approach focuses on the construction of execution models with detailed information about the start-up of the software components, which the organization calls Process Framework (PF) Clients. As part of the reconstruction, an iteration was necessary to validate the mapping rules and the output of the task identification activity. Fig. 8 shows one of the execution models that were constructed to zoom in on details. The model describes the part of the start-up process that happens in the Host computer, which is mainly the start-up of the software components within the Background Processes and Foreground Processes tasks described in the overview model. As we described before, these are simplified models. Fig. 8 does not illustrate the start-up of about other 40 software components that take less than 0.2% of the total start-up time.

The analysis of the models led to identify concrete evidence about the identified issues and opportunities for improvement. For example, the stars in Fig. 8 show that the start-up of the PF Client DBServer was quite long, which according to its function was not correct. Due to this situation, the PF Client Recon and PF Client Scanner task were delayed. We also identified that the Foreground Processes task was taking a long time mainly due to the start-up time of the PF Client GyroView and PF Client Examcards. With this concrete information at hand, we selected experts of the teams designing, developing, and maintaining the involved PF clients. Then, we established communication with each expert to discuss the execution models that describe the related issues, and to analyze the feasibility of development activities to address the issues.

4.4. Phase 4: Transition

The constructed execution models made several characteristics of the start-up process explicit and supported the communication of the identified issues. However, coordinating and starting development activities to implement solutions for the identified issues are not straightforward activities. The challenge of this phase is to provide detailed information that point out the implementation elements that are root causes of the mismatches and issues, but without being overwhelmed by the component or system implementation size and complexity.

For the start-up case study, this phase focused on the recovery of execution information that provides detail about an identified issue and the involved software components (PF Client). The required level of detail depended on the

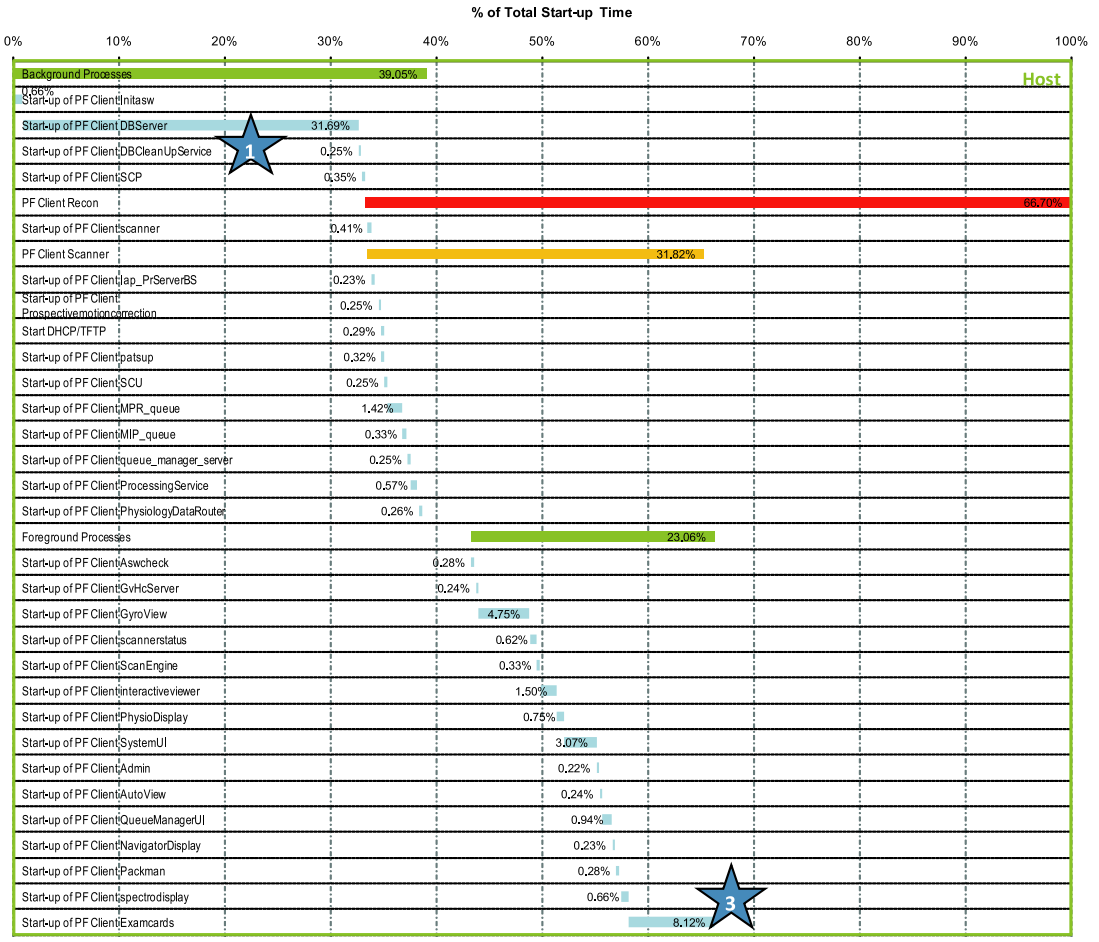

Fig. 8. A detailed as-is model of the system start-up in the Host computer.

knowledge and experience of the developers responsible for the involved PF Client. Detailed information can be identified by decomposing the start-up of the PF Client into finer tasks. An alternative was profiling, i.e., measuring the execution periods of code modules or functions that implement the involved PF Client. However, for some of the involved PF Clients, this was not an option due to a number of reasons. The collected runtime data, the practitioner's knowledge, or the PF Client implementation size and complexity did not provide accessible information to immediately implement profiling. Therefore, we decided to customize the dynamic analysis and experiment with a new interpretation and representation of the already available runtime data.

In the customization, we analyzed the start-up of a PF Client as series of execution periods. An execution period is defined by the time period between two consecutive logging messages collected for the execution scenario, excluding those periods with zero seconds and milliseconds. We consider that this customization of the technique is a kind of profiling, but at a different level of abstraction from profiling implementation elements. The construction of the execution model activity focused on creating a graphical representation that visualize long execution periods as peaks in the start-up of a software component. This allowed us to present detailed information without additional instrumentation or increasing the size and complexity of the runtime data.

Fig. 9 shows the constructed graphical representation to analyze the start-up of the PF Client Examcards, identified as an issue in the previous phase. In the figure we compared the start-up of the software component in three different execution scenarios of the system start-up, e.g., cold start, warm restart, and foreground restart. These scenarios were selected to determinate whether some of the peaks can be attributed to the availability of runtime platform or network services. The analysis of the representation helped us to conclude that the root cause of the long start-up of the PF Client Examcards was a connection timeout. With this information at hand, the involved architect was able to coordinate and start a downstream development activity to implement a solution to fix the time out situation. Other similar activities were coordinated and started to implement solutions for the other identified issues. The results of these activities led to considerable improvement of the system start-up, which the development organization considered as the main benefit of the case study.

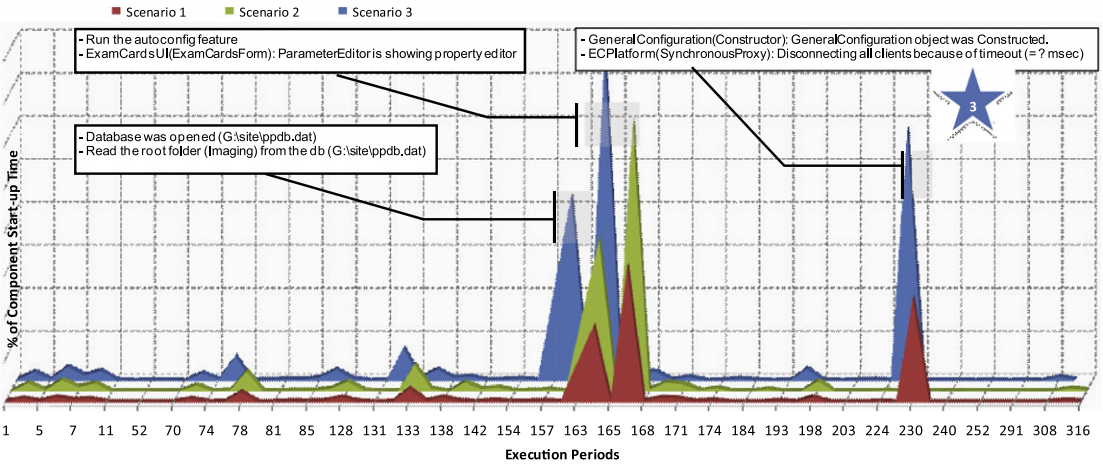

Fig. 9. A detailed description of a software component start-up.

5. Technical contribution

The SWAT highlighted that the contribution of the case study was the identification of quick wins to improve the system start-up time without being overwhelmed by the size and complexity of the system. The term *quick wins* refers to the relatively cheap and easy initiatives that were quickly designed and implemented. We identified that the contribution was possible because the strategy supported top-down analysis, communication between stakeholders, and the pragmatic description of runtime concurrency and performance. In practice, we observed that these aspects are key ingredients for improving the ability of large and complex systems to respond effectively to change.

5.1. Top-down analysis

Top-down analysis is a common practice to understand software systems. At an architecture level, top-down analysis is important to manage complexity by looking at the "big picture" first, and then to identify and analyze the details that matter for the problem at hand. Top-down analysis is an important practice to support architecture-centric evolution and downstream development processes. At Philips Healthcare MRI, the Software Architecture Team (SWAT) strives to implement an architecture-centric evolution for the Philips MRI scanner. The SWAT is responsible for the general architecture and design of the system. In the downstream process, the system is decomposed into several subsystems and components. Each subsystem and component is the responsibility of software designers and a development team.

The different roles and responsibilities in a downstream development process demand top-down analysis to address different information requirements. For example, while architects are mostly interested in high-level or architectural information, the programmers in a development team are interested in information about code or implementation constructs such as modules, libraries, and function calls. These information requirements were supported across the phases of the strategy. At the inception and elaboration phases, we constructed high-level descriptions based on the practitioners' knowledge (see Fig. 6) and the actual realization of the system (see Fig. 6). In the issue analysis and transition phases, we constructed a set of as-is execution models, including the ones in Figs. 8 and 9, to present detailed information, analyze a set of issues, and then to communicate with the related development teams.

5.2. Communication between stakeholders

Communication between stakeholders is important, especially to make sure that the requirements and the system are understood at the appropriate level of detail by stakeholders and implementers. Communication between stakeholders was established and supported with the tangible and updated evidence, i.e. execution models constructed through the phases of the strategy. In Section 4, we described how two types of communication links were gradually enabled through the four phases of the strategy. First, the technical expert and the architect started sharing technical knowledge about the characteristics of the system start-up process. Second, these two practitioners were able to identify another set of designers and developers, with whom they shared and discussed the findings from the issue analysis phase (see Section 4.3).

In both types of communications, the constructed execution models served as tangible evidence that triggered and supported discussions about the system start-up process. As the result of the established communication, the coordination and execution of downstream development activities. These development activities reduce the total system start-up time about 30% without altering the original characteristics of the operational level. The details of the improvements included an initial reduction of 25% addressing the first issue, the long start-up of the PF DBClient Server component. Later, the reduction reached 28% addressing the timeout issue in the start-up of the PF Client Examcards component. Finally, addressing the third

issue, changing the mechanism used by the feature of the Copy Recon Software task, the reduction reached about 30% of the original start-up time.

In addition to this set of communication links, nowadays the constructed execution view is helping the communication and dissemination of knowledge about the system start-up across the various development teams and providers of the development organization.

5.3. Runtime concurrency and performance

In our previous work, we described how the architecture reconstruction approach is used to construct views that describe runtime aspects such as runtime dependencies [10] and resource usage [11]. In this case, the strategy has enabled the construction of another view, which describes the runtime concurrency and performance of a system key feature at an architectural level. The view is now a benchmark to control and assess changes in the concurrency and performance of the system start-up. Currently, the view is reconstructed automatically to contribute in the system integration and regression tests, especially to monitor the redesigning of the integration process and to keep the system start-up performance within acceptable boundaries. Finally, the organization considered that an additional contribution of the case study is the setting of a precedent that supports the use of the top-down strategy to construct execution views and benchmarks for other key runtime features.

6. Related work

In this section we discuss work related to our results and contribution.

6.1. Architecture reconstruction

Our research and the contribution of this report is related to reverse engineering, especially architecture reconstruction solutions, i.e. methods, techniques, and tools that recover architectural information from existing systems [4–7]. Symphony [18] is a representative conceptual framework and an amalgamation of common patterns and best practices for architectural reconstruction. On the one hand, our work is an implementation and specialization of this framework, especially on two aspects. First, Symphony specializes reverse engineering using architectural views and viewpoints. We implement this by defining and applying execution viewpoints [9,14], including an execution metamodel, to construct execution views. Second, Symphony defines two explicit major phases to construct a required view, reconstruction design and reconstruction execution. We implement reconstruction design mainly at the inception phase and reconstruction execution through the other phases iterating with the reverse architecture reconstruction approach. On the other hand, our work shows how Symphony can be applied and embedded as part of a downstream development process.

6.2. Hypothesis-driven approaches

Hypothesis-driven architecture reconstruction approaches use a hypothesis model to guide the reconstruction process. The reflexion model by Murphy et al. [19] is the most prominent hypothesis-driven approach [4]. This approach summarizes a source model of a software system from the viewpoint of a particular hypothesis, i.e. a high-level model, specified by a practitioner. In the literature [4,19], one can distinguish two main characteristics of existing hypothesis-driven approaches. First, the specification of the hypothesis model that guides the reconstruction process corresponds to the implicit perception of a engineer or developer. Second, the hypothesis model and the constructed models correspond to module or implementation viewpoints [4].

The top-down strategy can be distinguished from existing hypothesis-driven solutions as follows. First, the goal of the top-down strategy is the construction and use of a high-level model for the runtime behavior rather than the implementation of a system. Second, the use and construction of the hypothetical view is mainly to support the goal of the inception phase, the elaboration of the construction requirements. Moreover, the construction and use of the hypothetical view follows the guidelines of a set of pre-defined execution viewpoint, which enables other practitioner to understand the information presented in the hypothesis. Besides this difference, hypothesis-driven solutions such as the reflexion model [19] and the top-down strategy in our work have in common the use of an explicit high-level model, the potential to compare this model with an actual or implemented model, as well as the use of mapping, i.e. mapping rules, to extract high-level information.

6.3. Profiling and visualization tools

Profiling and visualization tools are specific types of tools that are related and can contribute to our work. While profiling tools can help us to collect particular runtime data, visualization tools could improve the presentation and analysis of runtime information. On the one hand, profiling tools such the Eclipse Performance Tools Platform (TPTP) [20], YourKit [21], and AQTime [22], to name a few, collect runtime data as a software system or application executes. The runtime data includes memory and processor usage, the code elements that were executed, and the time that the application spends in each of the

code elements. On the other hand, visualization tools implement mechanisms to summarize [23] and condense [24] large amounts of runtime data, so it can be presented and analyzed as high-level information. In our work, the use of profiling and visualization tools is a problem-driven decision that takes into account the following aspects:

- *The need of global understanding*: Practitioners need a global understanding of the various aspects of the large and complex software-intensive systems they develop. Global understanding about the runtime behavior cannot be achieved by looking at every single code element. Instead, practitioners need to look at more coarse-grained abstractions. Profiling and visualization tools support it to some extent. Most profiling tools can collect runtime data for different abstractions of the implementation, e.g., methods, classes, and modules. At the same time, visualization tools can present the collected data to an even higher level of abstraction, which is useful to get a global understanding of the runtime of implementation abstraction. We first focus on mechanisms and tools that collect runtime data about higher and system-specific runtime abstractions (e.g., execution scenarios, tasks, components, processes), which reduces the need to abstract and present large amounts of runtime data. Then, if it is required, e.g. for issue analysis or transition phases, we can collect runtime data about implementation elements, either using monitoring or profiling tool, for specific and less complex parts of the system.
- *The heterogeneity of the system implementation*: While TPTP or YourKit can be used to collect runtime data for Java implementations, AQTime or YourKit for .Net implementations. For a large system with heterogenous implementation, like the software of the Philips MRI scanner, a combination of these various tools will be an ideal but expensive solution. Instead, we first focus on mechanisms and tools that gather runtime data independently of the implementation technology, i.e., logging and monitoring tools of the runtime platform.
- *Technical impact*: Once profiling is really required, the ideal profiling tool should be integrated smoothly into the development process. For example, we observed that practitioners developing large and complex software systems desire minimal changes in the source code, minimal overhead in the system response time, and a minimal learning curve to use tools. For this reason, the tool support for our reverse architecting approach (see Section 3.2) consists of tools and mechanisms that can be often found as part of a development and system infrastructure, such as logging, monitoring tools provided by the runtime platform.

6.4. Boot performance analysis tools

Regarding the system feature in our case study, our work is related to analysis tools such as Bootchart [25] and Bootvis [17]. Bootchart is a tool for performance analysis and visualization of the GNU/Linux boot process. The information that is analyzed includes resource utilization and process activity during the boot process. Bootvis allows designers and manufacturers to characterize their system's performance during Windows start-up to identify opportunities for performance improvement of Windows XP. The same aspect that we discuss for profiling and visualization tools drive the application of boot performance analysis tools. As we described in Section 4.2, using Bootvis was an option to address a part of the problem, the boot of the Host computer, but not the total start-up process of the Philips MRI scanner. This situation showed that scalability is another aspect that should be considered to select and use tools within the development of large and complex software-intensive systems. The scale of our problem includes the boot of the three computers that build the Philips MRI scanner (see Fig. 1) and the MRI software component distributed across them. Using boot performance analysis tools on each of the system computer and then combine the collected data can be an option to address the problem. However, reaching the desired level of abstraction and mapping the collected data to the system terminology will be hard compared to our approach.

7. Discussion and open issues

The top-down strategy enables the systematic construction and use of execution views within the incremental development of the Philips MRI scanner. In this section, we discuss some critical aspects and open issues that may influence the use of the top-down strategy in another context.

7.1. Proper viewpoints and metamodel

The execution viewpoints and the metamodel play a critical role providing guidelines and blueprints to guide the construction and use of execution models. Our experience with the top-down strategy shows that when these elements are available and adopted by a development organization, the construction of execution models can be a straightforward activity. However, having the proper execution viewpoints and metamodel for a large and complex software-intensive system demands good understanding about the various system stakeholders and their concerns, the characteristics of the system, and the development organization. Therefore, in a context where the system and the organization match closely the characteristics of the Philips MRI scanner and its development organization, our execution viewpoints and execution metamodel will enable the use of the top-down strategy.

7.2. Availability of sources of runtime information

The availability of sources of runtime information, especially logging files, is necessary to achieve the goal of the elaboration, analysis, and transition phases of the strategy. Logging is a feature often implemented as part of large software systems to record and store information of their specific activities into dedicated files. This is a common practice to support activities such as testing, debugging, and corrective maintenance of fairly large and complex systems. An important aspect to construct execution views is that logging should contain data that describe the workflow of the system functionality. In our previous work [10,11], we have demonstrated that this is important to achieve two technical aspects of our approach. First, high-level information, e.g. tasks and software components, can be extracted in terms of the system domain. Second, the extracted high-level information can be top-down synchronized and complemented by finer runtime information, e.g. runtime measurements collected by monitoring tools of the system runtime platform. Therefore, in a context where the system at hand is fairly large and complex, logging will be available and the application of the top-down strategy to use our architecture reconstruction approach will typically be feasible.

7.3. Mapping rules and domain knowledge

Mapping rules and domain knowledge are key elements to achieve two technical aspects for constructing and using sufficiently accurate execution models. First, mapping rules enable the direct extraction of high-level information at first and then detailed information, when it is needed. Second, the extracted information is in terms of system-specific names and abstractions. The latter reduces the need of detailed information to construct and analyze execution models. Defining mapping rules requires considerable knowledge about the system realization, e.g., the syntax and semantic of logging messages. This knowledge can be derived by automated process [26], or recovered from experts. In our work, we support the latter providing a schema and a repository to represent and store mapping rules. We use these resources to make domain knowledge that resides in data produced by the system at runtime and in the minds of technical experts accessible. Although identifying the knowledge, i.e. patterns, to define mapping rules and maintaining a repository are extra activities for practitioners, we learned that it worths the effort, especially to facilitate and speed up subsequent architecture reconstruction activities.

After this case study, we constructed more execution views of the system start-up for other software releases of Philips MRI scanner. Extracting the information for new execution models was an automatic activity reusing the defined mapping rules. Therefore, in a another context, the result of applying the top-down strategy and the architecture reconstruction approach would require the availability of a domain expert and the adoption of a mechanisms like mapping rules, which we consider feasible. Having a technical expert as part of a reverse architecting activity is often possible when the problem at hand is important for the development organization. Having more than one expert will be even more beneficial, but in the case of conflicts in perceptions between various experts, additional analysis and communication cycles would be needed in each phase of the strategy. Finally, the concept and use of mapping rules are common practice among architecture reconstruction [4].

7.4. Open issues

The open issues in our work include the balance between domain knowledge and design notations in the constructed execution model. The notations used in the constructed models, e.g., Figs. 6 and 8, are strongly coupled to the information requirements for the cases study and to the system domain knowledge. These factors contributed to achieve a quick and global understanding of the start-up process of the Philips MRI scanner. However, we have to point out that the constructed view may not be immediately used and understood by outsiders of the case study as well as of the development organization, who are not familiar with the corresponding execution viewpoints [9] or the system domain. This situation represents a barrier to incorporate the constructed execution view as a semi-formal or standard specification of the system.

A precise case where we need to improve the use of semi-formal or standard specification is the description of control and data flow that trigger temporal relations between task. Control and data flow are not explicitly presented in the constructed models, e.g. Figs. 6 and 8. Though, this information was not required through the inception phase, during the issue analysis and transition phases, this information was locally discussed to analyze the rationale behind concurrency and time periods. The first situation showed that keeping implicit description of control and data flow was good enough for architects and designers to achieve global understanding. The second situation showed that other practitioners need explicit description of control and data flow, especially to analyze runtime concurrency at a finer level of abstraction. Therefore, supporting the extension or transformation of the constructed execution models into more standard notations such as UML sequence diagrams and collaboration diagrams is an option, especially to benefit more stakeholders with up-to-date and accessible information about the system runtime structure and behavior.

8. Conclusions

This report described our experience with a top-down strategy for embedding the construction and use of execution views in practice. While the approach to construct execution views was developed and validated in our previous

work [10,11,9], this report describes the details of a systematic construction, use, and benefit of up-to-date execution views. The systematic construction was embedded in the downstream development process of a large and complex system. The use and benefit of the execution view enabled performance analysis and improvement of a system key feature. In addition, the report gives enough details to guide the reuse of the top-down strategy and reverse architecting approach in another context.

This experience is based on a particular development organization and its system, which may create some bias and threaten the validity and generalization of the results. Additional case studies with other similar systems and development organizations may be necessary to generalize our perception of the value of the top-down strategy and our architecture reconstruction approach in practice. We consider that this experience report is a reference for researchers and practitioners. For researchers, it provides insights about the practical or technical issues that need to be addressed when designing a process to use or embed reverse engineering solutions in a development organization. For practitioners, it shows the benefits of applying architecture reconstruction and specially how to support downstream development activities coping with the size and complexity of a software-intensive system like the Philips MRI scanner.

Acknowledgements

We would like to thank the Software Architecture Team and the software designers of the MRI scanner in Philips Healthcare. We also extend our gratitude to Pierre van de Laar and our other Darwin colleagues for their feedback and joint work.

References

[1] D. Garlan, D. Perry, Introduction to the special issue on software architecture, IEEE Transactions on Software Engineering 21 (4) (1995) 269–274.
[2] ISO/IEC Std. 42010: Recommended Practice for Architectural Description of Software-intensive Systems, 2007, http://www.iso-architecture.org/ieee-1471/.
[3] P. van De Laar, P. America, J. Rutgers, S. van Loo, G. Muller, T. Punter, D. Watts, The Darwin project: evolvability of software-intensive systems, in: Workshop on Evolvability at International Conference on Software Maintenance, 2007, pp. 48–53.
[4] R. Koschke, Architecture reconstruction: Tutorial on reverse engineering to the architectural level, Lecture Notes in Computer Science 5413 (2009) 140–173. doi:10.1007/978-3-540-95888-8_6.
[5] C. Riva, P. Selonen, T. Systä, J. Xu, A profile-based approach for maintaining software architecture: an industrial experience report, Journal of Software Maintenance and Evolution: Research and Practice 21 (2) (2009) doi:10.1002/smr.411.
[6] N. Sangal, E. Jordan, V. Sinha, D. Jackson, Using dependency models to manage complex software architecture, in: 20th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, ACM, 2005, pp. 167–176.
[7] A. van Deursen, Software architecture recovery and modelling, ACM SIGAPP Applied Computing Review 10 (1) (2002) 4–7. doi:10.1145/568235.568236.
[8] B. Cornelissen, A. Zaidman, A. van Deursen, L. Moonen, R. Koschke, A systematic survey of program comprehension through dynamic analysis, IEEE Transactions on Software Engineering 35 (5) (2009) 684–702. doi:10.1109/TSE.2009.28.
[9] T.B. Callo Arias, P. America, P. Avgeriou, Defining execution viewpoints for a large and complex software- intensive system. in: R. Kazman (Ed.), Joint 8th Working IEEE/IFIP Conference on Software Architecture & 3rd European Conference on Software Architecture, 2009, pp. 1–10.
[10] T.B. Callo Arias, P. Avgeriou, P. America, Analyzing the actual execution of a large software-intensive system for determining dependencies, in: 15th Working Conference on Reverse Engineering, IEEE Computer Society, 2008, pp. 49–58.
[11] T.B. Callo Arias, P. America, P. Avgeriou, Constructing a resource usage view of a large and complex software-intensive system, in: 16th Working Conference on Reverse Engineering, IEEE Computer Society, 2009, pp. 247–255.
[12] Philips Healthcare, Magnetic Resonance Imaging, 2010, http://www.healthcare.philips.com/main/products/mri/systems/.
[13] G. Muller, Conceptual View, 2004, Ch. 8, p. 81. http://www.gaudisite.nl/ArchitecturalReasoning.html.
[14] T.B. Callo Arias, P. Avgeriou, P. America, Tech. Report: Documenting a Catalog of Viewpoints to Describe the Execution Architecture of a Large Software-Intensive System for the ISO/IEC 42010 Standard, 2009, http://www.esi.nl/projects/darwin/publications.asp.
[15] Microsoft Corporation, Sysinternals Suite, 2010, http://technet.microsoft.com/en-us/sysinternals/.
[16] P. Kruchten, The Rational Unified Process: An Introduction, 2nd edition, Addison-Wesley, Boston, MA, USA, 2000.
[17] Microsoft Corporation, Fast Boot/Fast Resume Design, 2010, http://www.microsoft.com/whdc/system/sysperf/fastboot/default.mspx.
[18] A. van Deursen, C. Hofmeister, R. Koschke, L. Moonen, C. Riva, Symphony: View-driven software architecture reconstruction, in: Proceedings of the Fourth Working IEEE/IFIP Conference on Software Architecture, IEEE Computer Society, Washington, DC, USA, 2004, pp. 122–134.
[19] G. Murphy, D. Notkin, K. Sullivan, Software reflexion models: bridging the gap between design and implementation, IEEE Transactions on Software Engineering 27 (4) (2001) 364–380.
[20] Eclipse Test & Performance Tools Platform Project, 2010, http://www.eclipse.org/tptp/.
[21] YourKit . NET & Java Profiling, 2010, http://www.yourkit.com/.
[22] Code Profiler - AQtime, 2010, http://www.automatedqa.com/products/aqtime/.
[23] A. Kuhn, O. Greevy, Exploiting the analogy between traces and signal processing, in: Proceedings of the 22nd IEEE International Conference on Software Maintenance, IEEE Computer Society, 2006, pp. 320–329.
[24] B. Cornelissen, D. Holten, A. Zaidman, L. Moonen, J. van Wijk, A. van Deursen, Understanding execution traces using massive sequence and circular bundle views, in: 15th IEEE International Conference on Program Comprehension, IEEE Computer Society, 2007, pp. 49–58.
[25] Bootchart: Boot Process Performance Visualization, 2010, http://www.bootchart.org.
[26] Z. Jiang, A. Hassan, G. Hamann, An automated approach for abstracting execution logs to execution events, Journal of Software Maintenance and Evolution: Research and Practice 20 (4) (2008) 249–267.